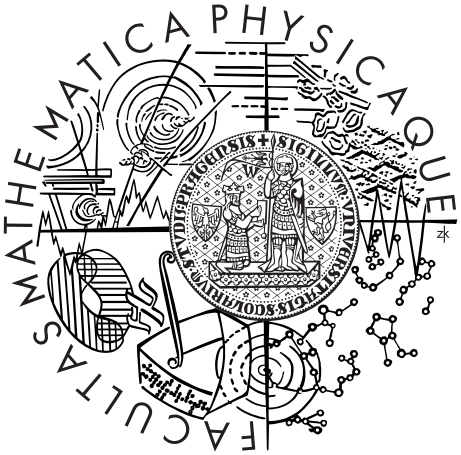
Charles University in Prague

Faculty of Mathematics and Physics

BACHELOR THESIS



Jan Michelfeit

Semantic Navigator

Department of Software Engineering

Supervisor of the bachelor thesis: RNDr. Tomáš Knap

Study programme: Computer Science

Specialization: Programming

Prague 2011

I wish to record my gratitude to my supervisor, RNDr. Tomáš Knap, for guidance he gave me throughout work on this thesis and also for his introduction to concepts of Linked Data.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague date .....

Název práce: Semantic Navigator

Autor: Jan Michelfeit

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Tomáš Knap

Abstrakt: Sémantický web rozšiřuje principy tradičního webu z dokumentů na data a umožňuje strojům porozumět významu informací. Internetové vyhledávače pro sémantický web toho mohou využít a nabízet přesnější výsledky vyhledávání a pokročilejší schopnosti dotazů. Cílem této práce je obohatit možnosti navigace mezi souvisejícími webovými stránkami s pomocí sémantických dat obsažených ve stránkách a zpřístupnit výhody vyhledávačů pro sémantický web běžným uživatelům. Jako řešení předkládáme nástroj Semantic Navigator implementovaný jako rozšíření do prohlížeče Mozilla Firefox. Pro nalezení souvisejících dokumentů využívá vyhledávací služby Sindice a poskytuje přístup k souhrnům informací na službě Sig.ma.

Klíčová slova: Sémantický web, navigace na Webu, sémantické vyhledávání, RDF

Title: Semantic Navigator

Author: Jan Michelfeit

Department: Department of Software Engineering

Supervisor: RNDr. Tomáš Knap

Abstract: The Semantic Web extends principles of the Web from documents to data. The Web of Data facilitates machines to understand the meaning of information. Semantic web search engines can take advantage of this fact and offer more precise search results and advanced query capabilities. The aim of this thesis is to use semantic data in web documents in order to enhance navigation between related web pages and bring advantages of semantic web search engines to ordinary users. We present Semantic Navigator as a solution to this task. Semantic Navigator is implemented as a Mozilla Firefox extension. The extension uses the Sindice search service for discovery of related documents and provides access to information summaries on Sig.ma.

Keywords: Semantic Web, web navigation, semantic search, RDF

Contents

Introduction

The richest source of information today is the World Wide Web. With increasing number of web pages, it may become more difficult to find the information we look for among many documents referring to similar search keywords. This thesis presents a tool that exploits the concept of the Semantic Web in order to enhance web navigation and help to overcome the keyword ambiguity problem.

The Semantic Web extends principles of the Web from documents to data. Unlike the contemporary Web, which is human-oriented, documents in the Semantic Web contain additional information with well-defined meaning understandable for machines. It creates an environment where software agents can carry out sophisticated tasks for users [9].

Traditional web search engines usually offer a keyword-based search. As mentioned above, one of the problems associated with this kind of search is the keyword ambiguity, i.e. one keyword has multiple meanings. The search engine cannot know the actual intended meaning. Another drawback is that it may be difficult to search for a property of an entity. The same property is expressed in different words in different documents. Search engines for the Semantic Web can cope with these problems better because they understand the meaning of what the user searches for.

This thesis introduces Semantic Navigator, a tool that utilizes structured data in web documents in order to bring advantages of semantic web search engines to ordinary users and provide an easy way to navigate between documents referring to similar or related concepts. Semantic Navigator is integrated in a web browser as an extension so as to provide a quick access to search functionality and simple interface for users not familiar with technical details of the Semantic Web. Furthermore, other features of the Semantic Web Stack, such as ontologies, are utilized.

Structure of the thesis. The thesis is organized as follows: The first chapter provides an overview of the technologies of the Semantic Web and introduces terms used in the following chapters. In Chapter 2, we present the motivation behind Semantic Navigator and describe both functional and non-functional requirements for the tool. Chapter 3 is devoted to the user interface. Chapter 4 covers the implementation, used components and documents design choices. Possible improvements and extensions are suggested in Chapter 5 together with several potential future applications. Chapter 6 compares Semantic Navigator to other projects with a similar purpose, before we summarize achieved results in Chapter 7.

Contents of the CD attached to this thesis are listed in Appendix A. Installation instructions and system requirements can be found in Appendix B. Finally, Appendix C gives an overview of source files and explains how to build the extension from sources.

1. Terminology and Technology Overview

The Semantic Web is a web of data that can be processed directly or indirectly by machines [8]. It is an abstract concept that needs an implementation. This chapter provides an overview of some languages and technologies used to create the Semantic Web. The hierarchy of languages where each layer uses capabilities of the layers below is also known as the Semantic Web Stack.

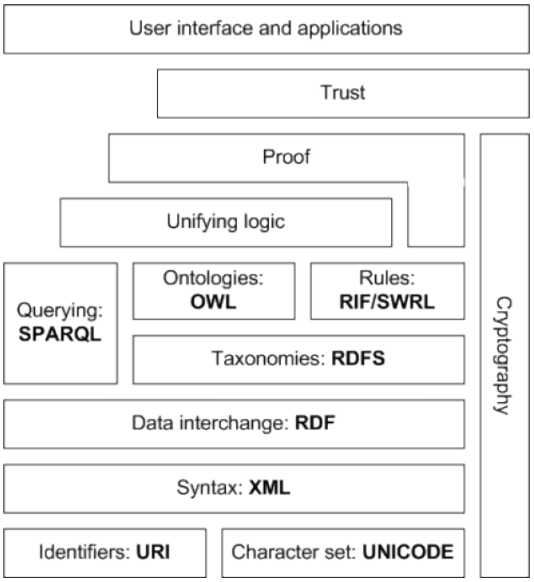


Figure 1.1: Illustration of the Semantic Web Stack
Source: www.obitko.com

1.1 RDF

Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web [24]. Its specification issued by W3C provides data model and methods for modeling information also about things than can be *represented* on the Web. RDF data are primarily intended to be used directly by applications and a common representation enables processing of data by different programs than just those the data were designed for.

*URI*s (Uniform Resource Identifiers) are used to identify RDF resources. Each resource can be described by a collection of typed statements in the form *subject-predicate-object*.

Every such *triple* describes a property of an RDF resource – the resource being described is in place of the subject, the property is used as the predicate (the terms *property* and *predicate* are often interchangeable), and the value of the property is in place of the object. A subject can be either a URI resource or an auxiliary *blank node*. A blank node differs from an RDF resource in that it does not have any URI assigned. An object can be a URI resource, a blank node or it can be a literal.

RDF triples compose a directed graph where subjects and objects are the nodes and predicates represent edges from the respective subject node to the object node. This data model is the basic instrument for representation of information in the concept of Semantic Web. Its fundamental advantage over the regular Web is meaning of information understandable for applications.

Shorthand URI notation known from XML namespaces is commonly used. A URI is written as a qualified name consisting of a prefix followed by a semicolon and a local part. Each prefix is assigned a URI. A qualified name is equivalent to concatenation of the prefix URI and the local part. Prefixes used in this thesis can be converted to corresponding URIs using prefix.cc.[1]

SPARQL is an RDF query language [28]. RDF graphs can be queried with SPARQL queries consisting of triple patterns, logical operations, value filters, optional patterns and other constructs. We can say that SPARQL is to the Semantic Web what SQL is to relational databases.

Alternatives to RDF

There are other methods how to describe meaning of resources and their properties on the Web besides RDF. They include microformats, microdata or eRDF; for more information see the following sections. Data represented in these formats can be converted to RDF data. Hereafter, we will refer to RDF data and data convertible to RDF as *semantic data* or *structured data*.

1.2 Linked Data

The term *Linked Data* refers to a set of best practices for publishing and connecting structured data on the Web [10, 6]. These practices can be summarized in the following principles:

- Things are identified with Uniform Resource Identifiers (URIs)
- The URIs used are dereferencable HTTP URIs

[1] `http://prefix.cc`

- When such an URI is looked up, useful information is provided using standard formats (e.g. RDF/XML)
- Links to other URIs are provided, so that related information can be discovered

Since Linked Data build on well-known technologies of the current Web like URIs and HTTP (HyperText Transfer Protocol), Linked Data is a natural extension of the existing Web and together they compose one large information space. Links to other URIs are represented by RDF triples where the subject is a URI reference from one data set and the object is a URI reference from another data set.

The principles of Linked Data have been adopted by the Linking Open Data project[2]. This community project has published a lot of data in various interlinked open data sets.

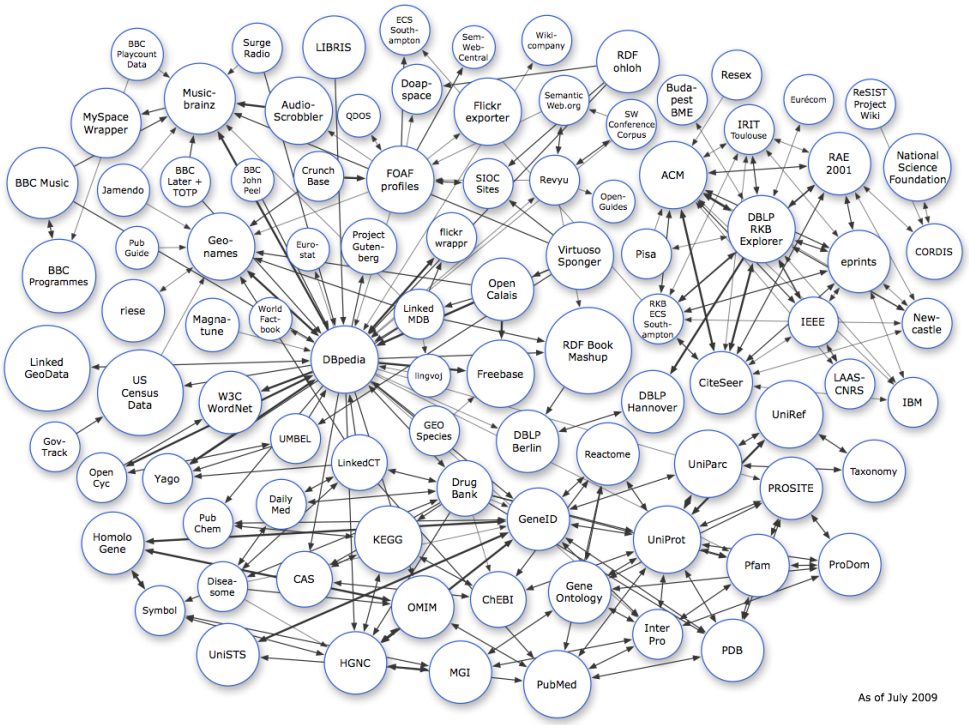


Figure 1.2: Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. The latest version can be found at `http://lod-cloud.net/`.

1.3 RDF Serialization Formats

RDF data can be serialized in several formats. This section lists the most common formats.

[2] `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`

RDF/XML is the normative RDF serialization format based on XML [24, 3]. The format is often called simply RDF because it was introduced among other W3C specifications defining RDF. An example is given in Listing 1.1.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:ex="http://example.org/stuff/1.0/">
  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
      dc:title="RDF/XML Syntax Specification (Revised)">
    <ex:editor>
      <rdf:Description ex:fullName="Dave Beckett">
        <ex:homePage rdf:resource="http://purl.org/net/dajobe/" />
      </rdf:Description>
    </ex:editor>
  </rdf:Description>
</rdf:RDF>
```

Listing 1.1: Example of RDF/XML

Notation3 is an RDF serialization format which provides a compact and lucid alternative to RDF/XML. Notation3 is extended to allow greater expressiveness compared to RDF/XML [5]. Listing 1.2 provides a brief example.

Turtle is a serialization format of RDF triples compatible with a subset of Notation3 [4].

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.
<http://en.wikipedia.org/wiki/Tony_Benn>
  dc:title "Tony Benn";
  dc:publisher "Wikipedia".
```

Listing 1.2: Example of Notation3

RDFa is the standard way for embedding of RDF data directly into an XHTML document [2]. Attributes of elements are used to define RDF triples. They include both standard XHTML attributes (e.g. `rel`, `href`) and attributes newly defined by RDFa specification (e.g. `resource`, `property`), as shown in Listing 1.3. This approach enables reuse of existing documents and avoids unnecessary content duplication. In addition, it enriches XHTML documents with useful metadata. Work on specification how to embed RDF data in plain HTML is currently in progress [1].

GRDDL is a markup syntax used to declare that an XML document contains data compatible with RDF and to link to algorithms for their retrieval (typically as XSLT) [15].

Embedded RDF[3] (eRDF) is a form of HTML that provides a way how to extract RDF

[3] `http://research.talis.com/2005/erdf/wiki/Main/RdfInHtml`

data from HTML documents. The approach is similar to RDFa serialization but eRDF does not introduce any new attributes and is only able to express a subset of RDF constructs.

```
<div about="http://dbpedia.org/resource/Albert_Einstein">
  <span property="foaf:name">Albert Einstein</span>
  <span property="dbp:dateOfBirth" datatype="xsd:date">1879-03-14</span>
  <div rel="dbp:birthPlace" resource="http://dbpedia.org/resource/Germany">
    <span property="dbp:conventionalLongName">Federal Republic of Germany</span>
  </div>
</div>
```

Listing 1.3: Example of RDFa

1.4 Microformats

Microformats represent yet another way how to embed structured data into an (X)HTML document. They make use of existing (X)HTML attributes, most notably the `class` attribute, to denote meaning of a piece of text in the document. Microformats are intended to lower the barrier of semantic markup for developers [18]. Mappings of microformats vocabularies to RDF are provided. Microformats are recognized by major search engines.

The fundamental difference from RDF is that microformats vocabulary is limited to a few predefined formats defined by the microformats community, for example *hCard* for contact information or *hCalendar* for events. A microformat is embedded in a document by setting the `class` attribute of an element to so called root class name denoting the microformat. Properties of the microformat are represented by nested elements. Their `class` attribute specifies the property. Textual content or an attribute (e.g. attribute `href` of element `<a>`) represents the respective property value. Properties can be nested.

```
<div id="hcard-Albert-Einstein" class="vcard">
  <span class="fn n">
    <span class="given-name">Albert</span>
    <span class="family-name">Einstein</span>
  </span>
  <div class="org">Institute for Advanced Study</div>
</div>
```

Listing 1.4: Example of the hCard microformat

1.5 Microdata

Microdata[4] is one of the suggested features of HTML 5. Microdata annotates the DOM[5] with scoped name/value pairs from custom vocabularies. It is easy to embed in a document like microformats but they are extensible like RDFa.

Every vocabulary defines a set of named properties. Each property is assigned a URI in the vocabulary namespace. Properties are not defined formaly (in contrast to RDF ontologies) but there is usually a human-readable description located at the vocabulary namespace URI.

A vocabulary is specified for a DOM subtree with a newly introduced attribute `itemtype` (see Listing 1.5). Attribute `itemscope` denotes a DOM node to which nested properties apply. Properties are specified with the `itemprop` attribute and the respective value is either textual content of the element or value of an attribute (similar to microformats).

Major search engines support Microdata and they provide vocabularies for several common types such as Person or Event[6].

```
<div itemscope itemtype="http://data-vocabulary.org/Person">
  My name is <span itemprop="name">Bob Smith</span>.
  I live in
  <span itemprop="address" itemscope itemtype="http://data-vocabulary.org/Address">
    <span itemprop="locality">Albuquerque</span>
  </span>.
</div>
```

Listing 1.5: Example of microdata

1.6 Ontologies

An ontology describes meaning of things from a specific domain and relations among these things in a form that is understandable for computer programs. This description consists of a collection of *classes* and *properties*. Using a reasoner, one can infer new statements about things based on ontology description. Ontologies used with Linked Data are expressed in RDF using definitions from RDFS [13] and OWL [26].

RDFS stands for RDF Schema. It is RDF's vocabulary description language. RDF works with subject-property-object triples but it provides no mechanism for describing properties. RDFS defines a set of classes and properties that can be used to describe classes, properties and other resources. For example, RDFS defines classes `rdfs:Class`,

[4] `http://www.whatwg.org/specs/web-apps/current-work/multipage/microdata.html`
[5] Document Object Model, `http://www.w3.org/DOM/`
[6] `http://schema.org`

`rdfs:Literal` and properties `rdfs:domain` or `rdfs:subPropertyOf` with self-explanatory names.

The *OWL Web Ontology Language* is a more expressive language used to define ontologies. It has three increasingly-expressive sublanguages (OWL Lite, OWL DL and OWL Full). OWL adds vocabulary for describing properties and classes. For example, classes can be declared as disjoint, properties can have a cardinality or they can be stated to be symmetric or transitive.

2. Specification

In this chapter, we discuss the use of structured data in navigation on the Web and the motivation behind Semantic Navigator. Then we cover both functional and non-functional requirements of the project.

2.1 Use of Structured Data in Web Search

The idea of Semantic Web was introduced by Tim Berners-Lee with a vision of intelligent software agents that would be able to retrieve information from the Web for users automatically [7]. Even though the idea dates back to the early history of the Web and a lot of progress in this area has been made, the use of structured data has not really penetrated web browsing experience of ordinary users. There are more reasons for this, e.g. the lack of universal software agents or that there are much less documents with structured data than there are traditional human-oriented ones.

Apart from these problems there is one more hindrance: Users are used to the traditional search paradigm – first search for relevant documents and then find information in these documents by going through them manually. However, before the shift to automated information retrieval can be made, the benefits of structured data can be used with the traditional search paradigm too.

2.2 Goals of Semantic Navigator

The main goal of Semantic Navigator is to make web navigation easier in discovering web documents with information

1. about a selected entity,
2. about relevant properties of the entity.

In the first case, the user can navigate to a web page about the film Casino, for example. If the user wants to find more information about the film, hyperlinks leading to related web pages can be followed or a web search engine can be used. Both approaches have their limitations. Hyperlinks lead only to documents that were known to the author of the website at the time of writing. Search engine results may not always be relevant because the word “casino” has other meaning than a film title. This problem is elegantly solved with the concept of Linked Data where all things are identified with unique URIs. Using a semantic web search engine, one can easily locate documents describing the film Casino.

However, because resource URIs are typically too long and not user-friendly[1], they are not directly visible for users. Thus a tool that would make use of these URIs for users is needed. Semantic Navigator provides a user-friendly way how to extract URIs from a web page and search related documents using these URIs.

The second point involves searching for a property of a resource. Another basic component of the Semantic Web Stack can be used here – ontologies. For instance, the user may want to know the release year of the film Casino. If a web page about the film contains information that a resource labeled "Casino" belongs to class "film" and this class has a property "release year" in the respective ontology, Semantic Navigator will let the user know about it and provide list of pages containing information about the release year.

More examples of how this approach can be used over PubMed[2] database of life sciences and biomedical topics are listed in thesis [19].

2.3 Support in Search Engines

In order to utilize structured data in document-oriented searches, we need support in web search engines. Major web search engines already parse some of the structured data published in web documents. So far they use it to provide more details in search results pages (see Figure 2.1). This feature is included in Yahoo! Search, Google Search (called *rich snippets*[3]), support in Bing is planned [30].

Figure 2.1: Yahoo! Search results with contact information (telephone and address) extracted from microformats

Specialized semantic web search engines index primarily semantically enabled pages. They are able to provide searches based on resource URIs and even resource properties in addition to keyword searches. Search engines can be categorized as human-oriented, such as Falcons [14] and SWSE [20], and application-oriented, such as Swoogle [22], Watson [16] or Sindice [27].

[1] E.g. URI `http://musicbrainz.org/mm-2.1/artist/00a9f935-ba93-4fc8-a33a-993abe9c936b` identifies music band Nightwish on MusicBrainz.

[2] `http://www.ncbi.nlm.nih.gov/`

[3] `http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=99170`

Search engines in the first category provide keyword-based search similar to traditional search engines. Resources in semantically enabled documents can have explicitly defined names (e.g. using `rdfs:label` property). That allows search engines to search directly in these names. In addition, search in Falcon can be refined by navigating a class hierarchy.

The second category, application-oriented search engines, is useful for applications built on top of search engines. While they may provide a keyword search too, search capabilities are usually extended to exploit the potential of semantic data. Resources can be looked up by their resource URI, search can be limited to certain classes or advanced searches using RDF triple patterns can be implemented. Furthermore, search results are available in a machine-readable format, often directly as RDF data.

2.4 Search Methods

In accordance with goals described in previous sections, search methods listed below are implemented. The Sindice service is used for all searches. More about the choice of Sindice can be found in Section 4.2.

2.4.1 Resource Search

The basic search method is resource search. List of documents that contain the resource URI is fetched from Sindice and presented to the user.

The user can also choose to display information about the resource gathered from multiple sources using Sig.ma.[4] See Section 4.2.1 for more information about Sig.ma.

2.4.2 Property Search

Another search method is property search. Documents that contain a *resource-property-** triple for the selected resource and property are returned.

In addition, the Sindice SPARQL endpoint can be queried for a list of possible values of the property. The query is essential for obtaining possible objects. Values present in RDF data extracted from the current document could also be used but when the document contains a complete *subject-property-object* triple, the object is usually displayed somewhere on the page and can be accessed directly.

2.4.3 Literal Search

A large number of triples embedded in (X)HTML documents contain a literal rather then an RDF resource as the object. RDF data represented by microformats do not allow any

[4] `http://sig.ma`

other type of objects. Literals can be searched with a keyword-based query, as in traditional web search engines.

Moreover, the search can be refined by selecting a property and searching for the triple pattern **-property-literal*. Justification of this type of search is illustrated with the following example. Say we have a triple with predicate `foaf:lastName` and object "Washington". Simple search for the literal value would include not only people but also cities named Washington. When we search for the literal as a value of `foaf:lastName`, the search will be limited only to people as `foaf:lastName` has its domain (`rdfs:domain`) set to class `foaf:Person` in its defining ontology FOAF[5].

2.4.4 Microformat Search

Microformat search is the last search method. Even though microformats are similar to blank nodes in RDF as they do not have an identifying URI, capability of Sindice to parse microformats can still be exploited. Chosen relevant values for each microformat are searched. For example, in case of the *geo* microformat, we can search for documents that contain the same values of `vcard:latitude` and `vcard:longitude` properties as the values in the microformat the user selected for search. Even though it cannot be guaranteed that the latitude and longitude values belong to the same microformat in result documents because only simple triple patterns with the `*` wildcard are supported by Sindice, support of more complex queries is planned. More about this topic is given in Section 4.5.2.

2.5 Non-Functional Requirements

Functional requirements for the project follow from what is covered in previous sections. This section lists other requirements that shall be taken into account.

First of all, use of the extension should be intuitive even for users not familiar with concepts of the Semantic Web. Furthermore, the whole process from selecting an initial resource or literal to getting search results should not be longer than searching manually using a web interface of a search engine.

Another requirement is that Semantic Navigator should not be too demanding in respect of system resources, especially when it is not being actively used. The Sindice service should not be overloaded with unreasonable number of queries.

And lastly, the extension should be multi-platform.

[5] `http://www.foaf-project.org/`

3. User Interface

The previous chapter is devoted to the specification of Semantic Navigator features. This chapter describes the project from a user's point of view. The user interface is an essential part of the project. It should be designed to be intuitive even for users not familiar with technical details of the Semantic Web, the user should not "get lost" in menus and it should lead the user to his or her objective (getting search results) quickly.

Semantic Navigator is integrated in Mozilla Firefox as an extension. The default language is English, other languages can be easily added.

While the user browses through the Web, Semantic Navigator shows its presence only with an icon in the browser statusbar. When it detects semantic data in a document, it is indicated by a change of the icon. The extension currently detects RDFa, microformats and external RDF documents linked to the (X)HTML document by a `<link>` tag.

Clicking the icon opens a popup menu. We will refer to it as *the statusbar menu*.

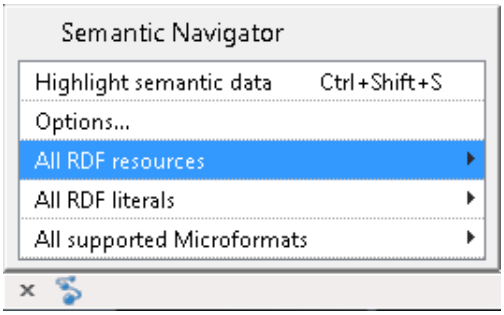


Figure 3.1: The statusbar menu. The "S" icon in the bottom left corner indicates presence of semantic data.

3.1 Highlighting Semantic Data in a Document

The statusbar menu contains an item named **Highlight semantic data**. As the name suggests, it highlights URI resources, microformats and RDF literals that are visible in the current document. A keyboard shortcut (`Control+Shift+S` by default) is also available.

A small icon appears next to highlighted elements as shown in Figure 3.2. The icon can be clicked in order to display a context menu. Contents of the context menu depend on whether it is opened for a resource, a literal or a microformat.

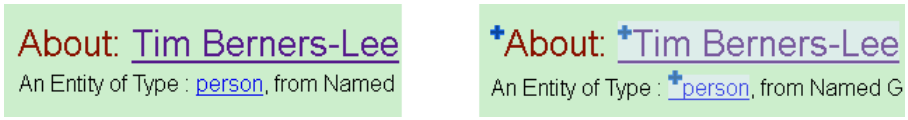


Figure 3.2: An example of highlighted elements in DBpedia article about Sir Tim Berners-Lee. The image on the left is the original appearance of the article, the image on the right shows the article with highlighted resources and literals.

3.2 Context Menus

All search methods are available through context menus. Because the user may navigate into multiple submenus during the process of searching, default submenus that appear next to their parent menus (used in Semantic Browser in Figure 6.1, for example) were replaced. Submenus appear in place of its parent menu instead. The user can navigate back and forth easily using a back button in the top left corner of the submenu. Furthermore, the menu can be comfortably navigated with arrow keys and the `enter` key. This way the user does not need to move mouse cursor around so much and does not have to worry about accidentally closing the submenu hierarchy.

3.2.1 Resource Menu

The context menu for a resource can be opened by clicking on a highlighted resource. A complete list of RDF resources is also available from the statusbar menu because not all resources may be visible on a web page.

The menu contains the following items (see Figure 3.3):

Search for documents containing the resource
: Displays a list of documents that contain the URI of the resource (resource search).

Search for the resource in Sig.ma
: Displays a summary of information about the resource in Sig.ma (see Section 4.2.1).

List of **Search for property** ⟨*property*⟩
: Searches documents that contain information about the selected property of the resource (property search).

List possible values of the property
: Displays a submenu with values of the selected property of the resource. The values are obtained both from data extracted from the current document and data fetched from a query to the Sindice SPARQL endpoint. Clicking on an item in this submenu displays list of documents that mention that value (resource search or literal search).

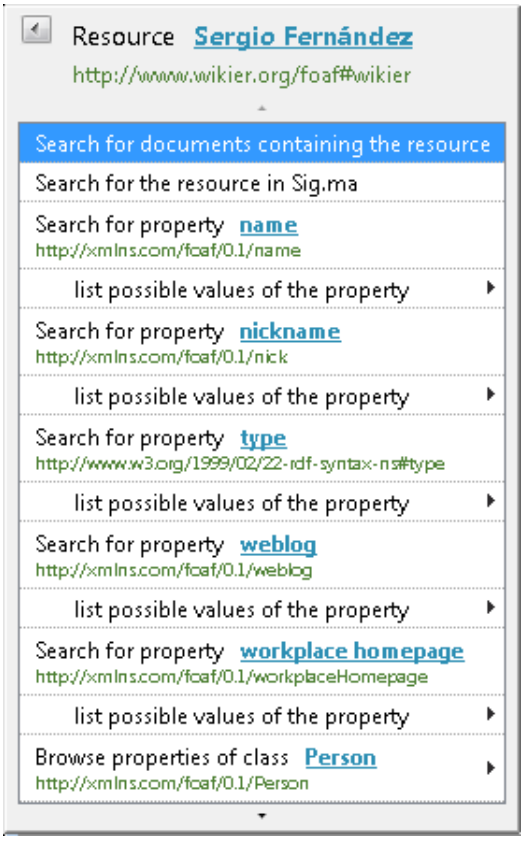


Figure 3.3: Context menu for a URI resource

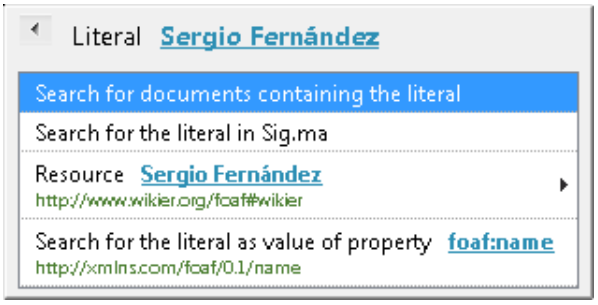


Figure 3.4: Context menu for a literal

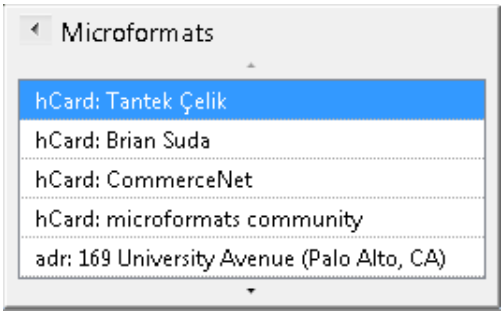


Figure 3.5: List of microformats in the statusbar menu

List of **Browse properties of class** ⟨*class*⟩
: Displays a submenu with a list of properties that have the selected class as its domain. Only classes that the resource in question belongs to are displayed. Items in the submenu are analogous to property items in the resource menu.

3.2.2 Literal Menu

Figure 3.4 shows the context menu for a literal. The menu can be opened by clicking on a highlighted literal or through a complete list of literals in the statusbar menu.

The menu contains the following items:

Search for documents containing the resource
: Displays a list of documents that contain the literal (literal search).

Search for the literal in Sig.ma
: Displays a summary of information based on a keyword search for the literal value in Sig.ma.

Resource ⟨*resource*⟩
: This menu item is available only for literals that represent a label of a resource. Selecting the menu item opens the context menu for the selected resource. This is particularly useful when resources are not defined in an (X)HTML document (and thus not visible there) but they are defined in a document attached by a `<link>` element and only RDF literals are highlighted in the (X)HTML document.

List of **Search for the literal as value of property ...**
: Displays a list of documents that contain a **-property-literal* triple pattern for the selected property and literal. Only properties that have an occurrence in such a triple in the RDF data extracted from the current document are displayed.

3.3 Search Results

Search results are presented in a new tab. An example is given in Figure 3.6.

Figure 3.6: Search results

3.4 Options Dialog

The options dialog can be opened from the statusbar menu. Here is the list of options:

Load external documents
If enabled, triples extracted from external documents attached to an (X)HTML document by a `<link>` element are added to the RDF model.

Load ontologies
If enabled, used ontologies are loaded and added to the current RDF model.

Cached ontologies
Maximum number of cached ontologies can be set here. Buttons for clearing the ontology cache and computing the total cache size are also available.

Use Java for loading external documents
If checked, external documents are loaded directly by Java. External documents are loaded by the web browser otherwise (slower, but does not require an exception in the firewall).

Search results
Results can be optionally sorted by date and displayed either in the current tab or in a new tab.

Accepted formats of search results
Accepted formats of documents returned in search results can be chosen here. For more information about reliability of format detection see Section 4.5.1.

3.5 A Real World Example

In this section we demonstrate how Semantic Navigator can be used on a real world example as shown in figures 3.7-3.12.

In our example, we begin at the homepage of Sergio Fernández[1] which is annotated with RDFa data. First we use Semantic Navigator to highlight the RDFa data (Figure 3.7). Then we select the resource representing Sergio Fernández (Figure 3.8). Say we are interested in people that Sergio Fernández knows. The relation "knows" is expressed by predicate `foaf:knows`. We can reach it by selecting the `foaf:Person` class (Figure 3.9) and the property is selected in Figure 3.10. Consequently, Semantic Navigator obtains the list of people Sergio Fernández knows (Figure 3.11) and we can search for more information about each person. The search results are shown in Figure 3.12.

[1] `http://www.wikier.org`. Sergio Fernández is the author of Semantic Radar described in Section 6.3.

Figure 3.7: Highlighted literal "Sergio Fernández"

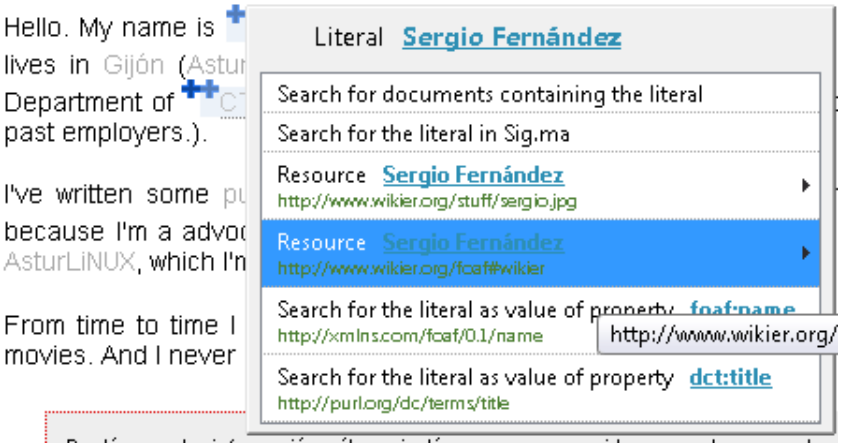


Figure 3.8: Selecting the resource representing Sergio Fernández

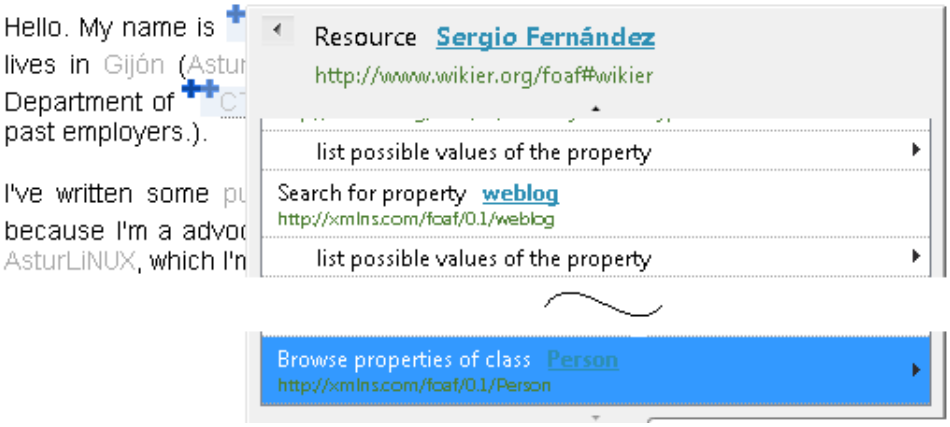


Figure 3.9: The resource representing Sergio Fernández belongs to the `foaf:Person` class

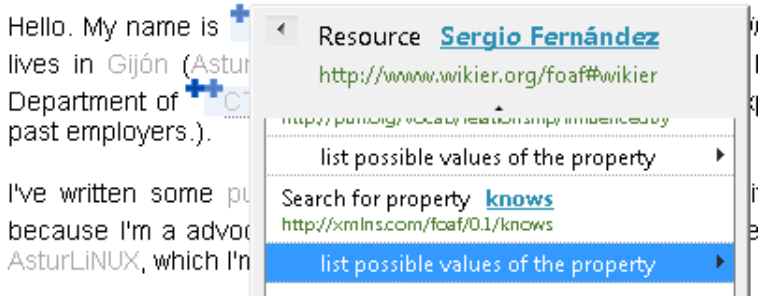


Figure 3.10: List of properties that have `foaf:Person` as their domain. For each property, we can search for documents containing a *Sergio Fernández-selected property-** triple or we can list objects of such triples.

Figure 3.11: List of values of the `foaf:knows` property of the resource representing Sergio Fernández

Figure 3.12: Search results for Chris Bizer

4. Project Design

This chapter covers the implementation of Semantic Navigator, components it uses and documents design choices that were made during development.

An outline of Semantic Navigator's architecture is given in Figure 4.1.

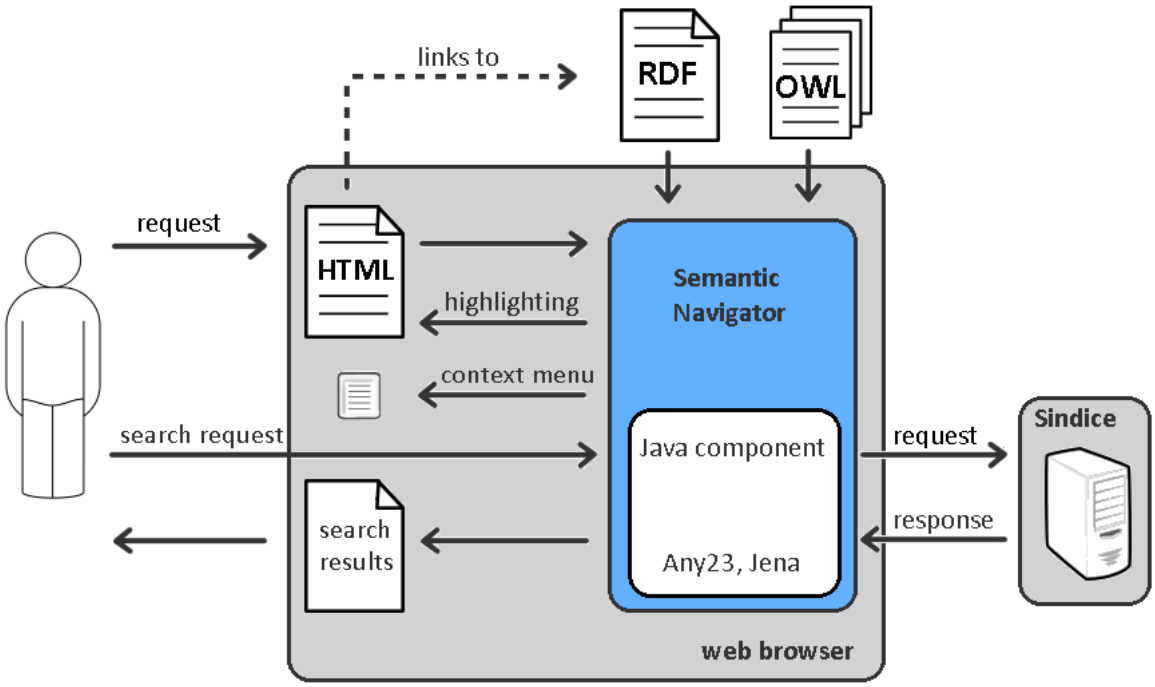


Figure 4.1: Basic components of Semantic Navigator

Semantic Navigator resides inside the web browser. When the user requests a HTML document with semantic data, Semantic Navigator processes it with Any23 and Jena Java libraries, optionally together with linked external RDF documents and used ontologies. Semantic data can be highlighted in the HTML document and accessed through context menus. The user can then perform a search based on the provided data. Search results are obtained from the Sindice service.

4.1 Mozilla Firefox Extensions

Since Semantic Navigator is intended as an aid in navigation on the Web, implementing it as a web browser extension is a natural choice. Mozilla Firefox is both multi-platform and offers a rich support for extensions. In addition, Java code can be called from extensions.[1]

[1] `https://developer.mozilla.org/en/java_in_firefox_extensions`

The use of Java enables utilization of existing Java libraries that will be discussed in Section 4.3.

The choice of Mozilla Firefox has implications about programming languages available for implementation. Mozilla Firefox extensions are written in JavaScript and XUL (XML User Interface Language). Even though libraries written in an arbitrary programming language can be used through the XPCOM model[2], such libraries would have to be compiled and distributed separately for each platform. Fortunately, Java is also available from extensions. Thus the selection of existing libraries is limited to JavaScript and Java in order to keep Semantic Navigator multi-platform.

More information about Mozilla Firefox Extensions can be found at Mozilla Developer Network.[3]

4.2 Search Engine

An overview of semantic web search engines is given in Section 2.3. *Sindice* is the only search engine that meets the requirements of Semantic Navigator both in query capabilities and a large updated index of data sources.

Sindice helps developers to deal with the decentralised nature of the Semantic Web by crawling and indexing data sources in the Semantic web. Its index is then exposed through a public API. The API provides for advanced queries for locating data sources. Queries can specify required or excluded keywords, resource URIs or RDF triples, the search can be limited by data source format, date, ontology, etc. Triple queries support a * wildcard that represents an arbitrary value in any part of a triple pattern.

Access to gathered triples through a public SPARQL endpoint has been recently added by the Sindice team [30]. The entire Sindice dataset with more than 12 billion triples can be queried through the endpoint.

4.2.1 Sig.ma

Sig.ma [31] is a project related to Sindice. It is a Linked Data browser rather then a search engine. Nonetheless, Semantic Navigator makes use of Sig.ma as a complementary service available to the user.

Sig.ma is a service that offers live, embeddable information summaries about entities. Sig.ma is powered by Sindice which it uses in order to find sources that mention the searched entity and show heterogeneous data aggregated from these sources. Since Sig.ma provides resource URI queries as well as a keyword-based search, the use of Sig.ma in Semantic

[2] `https://developer.mozilla.org/en/XPCOM`

[3] `https://developer.mozilla.org/en/Extensions`

Navigator benefits from the fact that Semantic Navigator can readily extract resource URIs from web documents and feed them to Sig.ma search. This feature is available through the resource or literal context menu. The user can also use Sig.ma from search results page as shown in Figure 4.2.

Figure 4.2: Sig.ma in search results

4.3 Used Libraries

This section covers requirements imposed on used libraries and describes reasons that led to the choice of Jena for manipulating RDF data and Any23 as a parser.

Semantic Navigator requires a means of representing and manipulating RDF data. Data for the RDF model are loaded both from (X)HTML documents in the browser window and from external documents. RDFa and microformats shall be parsed in the former, at least RDF/XML in the latter.

The Semantic Navigator extension also needs some basic query and reasoning capabilities. Direct support of ontology processing is favourable. At least the following types of queries are necessary:

- List all URI resources and literals
- List triples with a selected subject or object

- List triples with a selected subject and property
- List classes a selected resource belongs to
- List properties with a selected class as their domain

As stated above, the choice is limited to JavaScript and Java libraries. Even though parsers of various RDF serialization formats (RDFa, RDF/XML, Turtle) and query engines are implemented in JavaScript[4], none of available tools has support of ontologies and basic inference. For that reason, use of Java libraries, which provide broader possibilities, is necessary.

The *Any23* library[5] provides parsers of RDFa, microformats, RDF/XML, Turtle and Notation3. Any23 was chosen for its rich support or formats and because this library is used by Sindice too. That is a guarantee of some consistency with search results returned from Sindice, performance optimization and support in future.

There were two candidates for manipulation with RDF data and ontologies among available libraries:

Jena[6] is a framework for building Semantic Web applications. It is well-documented and both predefined reasoners and external reasoners can be plugged in. Disadvantages include a big size of the library or a complicated caching mechanism.

OWL API[7] provides an API focused on creating and manipulating OWL ontologies. The advantages are support of various reasoners and ontology caching. The API is actively developed and used in a number of projects (e.g. editor Protégé[8]). On the other hand, it is designed specifically for ontologies and not for manipulation with instance RDF data.

Jena was chosen from these two options on account of its good documentation and great flexibility that proved useful when plugging in Any23 parser and a custom cache management component.

4.4 Implementation Details

The Semantic Navigator extension operates in several steps that correspond to functional components of the extension. This section describes each of the components.

4.4.1 RDF Data Detection

The first step is detection of RDF data or data convertible to RDF (we refer to them as *semantic data*).

[4] A list of tools usable from JavaScript can be found at `http://www.w3.org/2001/sw/wiki/Javascript`

[5] `http://developers.any23.org/`, available under Apache License 2.0

[6] `http://jena.sourceforge.net/`, available under New BSD License

[7] `http://owlapi.sourceforge.net/,availableunderLGPL`

[8] `http://protege.stanford.edu/`

Semantic data present in the (X)HTML source of the current web page are detected first. RDFa and microformats are currently supported. Presence of RDFa is detected when at least one of the following conditions is met:

- Document DOCTYPE contains `-//W3C//DTD XHTML+RDFa 1.0//EN`
- Attribute `version` of element `<html>` has value `XHTML+RDFa 1.0`
- Any of the elements in the document has at least one of attributes `about`, `property`, `resource`, `datatype`, `typeof` or `instanceof` defined.
- Any of the elements in the document except for `<meta>` have attribute `content`

These conditions are based on the RDFa specification [2]. Microformats are detected using the native Firefox API.

Next Semantic Navigator attempts to locate external documents with RDF data. Supported formats are RDF/XML, Notation3 and Turtle. External documents referenced by a `<link>` element are detected. The `<link>` element must have attribute `type` with value "application/rdf+xml" for RDF/XML, "text/n3" for Notation3 or "text/turtle" for Turtle.

4.4.2 RDF Data Extraction

The next step involves RDF triple extraction from data sources identified in the previous step. Any23 is used both for the current (X)HTML document and external documents. An RDF model consisting of the extracted triples is created with Jena.

Human-readable labels for each resource that is displayed to the user is looked up if possible. Predicate `rdfs:label` is used preferably. Many ontologies, however, use their own properties to designate a label, most notably `foaf:name` and `dc:title` [21]. Because not all of them are defined as subproperties of `rdfs:label`, Semantic Navigator contains an explicit list of label properties. If no label is available, the shorthand URI notation or the full URI is displayed.

4.4.3 Importing Ontologies

Used ontologies are imported to the RDF graph if enabled in extension settings. This brings two benefits. First, properties are displayed with their label rather then URI. Second, properties of all classes a resource belongs to can be listed and searched for (see Browse properties of a class in Section 3.2.1).

Ontologies are fetched as described in [27]. Definition of each property is fetched by dereferencing its URI, following the Linked Data principle. Returned definitions are imported into the RDF graph. Furthermore, ontologies are recursively imported following `owl:import` links.

Again, ontology definitions are parsed with Any23 and imported as a Jena RDF graph.

Because loading ontologies every time Semantic Navigator is used would be very slow, a caching mechanism is needed. This idea is also supported by the fact that there are only a few commonly used vocabularies in the Linking Open Data Cloud [11]. Thus triples extracted from each ontology are cached locally with the least-frequently used replacement policy. Time required to process ontologies with caching proved to be reasonably short.

4.4.4 Highlighting Semantic Data in a Document

One of the key features of Semantic Navigator is highlighting of semantic data in web documents. The extension can highlight RDF resources, literals and microformats.

RDFa

First of all, data embedded as RDFa are highlighted. The process is based on the RDFa specification. All elements that have one of `src`, `href`, `about` or `resource` attributes holding an URI of a resource from the current RDF graph are highlighted. The URI can be expressed as a CURIE [2] in case of `about` and `resource` attributes. Then RDFa literals are looked up. They are represented by child nodes of elements with the `property` attribute.

XPath[9] is used to locate elements to be highlighted. Blank nodes are not highlighted because search by a blank node is not supported for the time being.

Microformats

Mozilla Firefox provides an API for managing and parsing microformats. This API is used to find and highlight currently supported microformats: hCard, geo, adr and hCalendar.

Literals in Text

The last step is highlighting of literals from the RDF model in the text of the current web page. This feature is useful particularly when semantic data are stored separately in an external document, referenced with a `<link>` element, rather then embedded in the (X)HTML document. It is one of the key features of Semantic Navigator (especially with regard to applications described in Section 5.2).

The literals are annotated using a modified Aho-Corasick string dictionary-matching algorithm [25]. The algorithm is described in pseudocode in listings 4.1 and 4.2. Details of Aho-Corasick are omitted since it is a well-known algorithm.

[9] `http://www.w3.org/TR/xpath/`

The search dictionary consists of all literals from the RDF model. Matching is executed on a word by word basis rather then letter by letter which would be an extra overhead in JavaScript. Only the longest leftmost match is annotated. The original Aho-Corasick algorithm is only capable of finding the longest *rightmost* match, however. For this reason, when a match is detected, it is not annotated immediately but stored in variable $lastMatch$ (line 17 in Listing 4.1). It can be extended when a longer match starting at the same position is found (line 14). The actual annotation takes place when we are sure that the match cannot be extended (another match is found on line 16 or end of input is reached on line 22). If an element is to be annotated both for a URI resource and a literal, URI resources take precedence.

The standard algorithm is designed for plain text. Therefore, adjustments for matching in (X)HTML were necessary. A match can span multiple nodes in the corresponding DOM tree but it should not contain block-level elements, which are semantically separated from the surrounding text.

The beginning and end of a match can be located in different depths of the DOM tree. A match is annotated by wrapping it in an extra element, thus all nodes that are part of a match must have the same parent element. Nodes with the same parent element are obtained on lines 8-10 of Listing 4.1 using the procedure described in Listing 4.2. The DOM tree is traversed up from nodes containing the first and the last word of a match until a common parent is found. First the same depth in the DOM tree is reached in lines 1-17 and second we continue until a common parent is reached in lines 18-20. The match can be extended at most with whitespace characters while traversing the DOM tree (lines 5 and 13) and block-level elements are avoided (lines 3 and 11).

This approach solves another annotation problem. Let us consider code `<b>nervous system</b> function`. Literal “system function” should not match because “system” belongs to “nervous” and should not be annotated without it. We can observe that the procedure from Listing 4.2 annotates words in a nested element, therefore related, only together.

Input: Set of literals *dictionary*, DOM node *root*.
Result: Occurrences of literals annotated in text content of *root*.

1. construct automaton: initialize goto and failure function, initialize output function $out(q)$ which gives the set of literals recognized in state q
2. $lastMatch \leftarrow$ **null**
3. $state \leftarrow 0$ // initial state
4. **for all** *word* in text contents of *root* **do**
5. $\quad state \leftarrow transition(state, word)$ // based on goto and failure functions
6. $\quad$ **for all** matched literals in $out(state)$ from the longest match to the shortest **do**
7. $\quad\quad firstWord,\ lastWord \leftarrow$ first and last word of the match in text
8. $\quad\quad$ **if** $firstWord$.DOMNode $\neq$ $lastWord$.DOMNode **then**
9. $\quad\quad\quad firstWord,\ lastWord \leftarrow getMatchInSingleNode(firstWord,\ lastWord)$
10. $\quad\quad$ **end if**
11. $\quad\quad$ **if** $firstWord \neq$ **null and** $lastWord \neq$ **null then**
12. $\quad\quad\quad currentMatch \leftarrow$ all DOM nodes between $firstWord$ and $lastWord$
13. $\quad\quad\quad$ **if** $lastMatch$ begins with $firstWord$ **then**
14. $\quad\quad\quad\quad lastMatch \leftarrow currentMatch$ // extension of the last match
15. $\quad\quad\quad$ **else if** $currentMatch$ does not overlap with $lastMatch$ **then**
16. $\quad\quad\quad\quad$ annotate $lastMatch$
17. $\quad\quad\quad\quad lastMatch \leftarrow currentMatch$
18. $\quad\quad\quad$ **end if**
19. $\quad\quad$ **end if**
20. $\quad$ **end for**
21. **end for**
22. annotate $lastMatch$

Listing 4.1: Literal annotation algorithm

Input: First and last word of a match $firstWord$ and $lastWord$.

Returns: First and last word of the match with their respective DOMNodes having a common parent. Span of the match can be extended at most with whitespace characters and must not contain a block element.

1. $minDepth \leftarrow min(depth(firstWord.\text{DOMNode}),\ depth(lastWord.\text{DOMNode}))$
   // $depth(node)$ returns the depth of $node$ in DOM tree
2. **while** $depth(firstWord.\text{DOMNode}) > minDepth$ **do**
3. **if** $firstWord$.parentDOMNode is block element **then**
4. **return** **null**, **null**
5. **else if** there are non-whitespace characters in previous siblings of $firstWord$ **then**
6. **return** **null**, **null**
7. **end if**
8. $firstWord.\text{DOMNode} = firstWord.\text{parentDOMNode}$
9. **end while**
10. **while** $depth(lastWord.\text{DOMNode}) > minDepth$ **do**
11. **if** $lastWord$.parentDOMNode is block element **then**
12. **return** **null**, **null**
13. **else if** there are non-whitespace characters in next siblings of $lastWord$ **then**
14. **return** **null**, **null**
15. **end if**
16. $lastWord.\text{DOMNode} = lastWord.\text{parentDOMNode}$
17. **end while**
18. **while** $firstWord$.parentDOMNode $\neq lastWord$.parentDOMNode **do**
19. repeat steps 3-8 and 11-16
20. **end while**
21. **return** $firstWord,\ lastWord$

Listing 4.2: Literal annotation algorithm: $getMatchInSingleNode()$

4.4.5 Search Results

Search results are obtained from Sindice and presented to the user in a new browser tab.

A query to the Sindice search service is a simple HTTP GET request, results are returned in JSON. The same goes for requests to Sindice SPARQL endpoint when querying for possible values of a property of a subject (see Section 2.4.2). Listing 4.3 shows the query.

```
SELECT DISTINCT ?object
WHERE { <subjectURI> <propertyURI> ?object. }
LIMIT maxResults
```

Listing 4.3: Query for obtaining property values from Sindice SPARQL endpoint. `subjectURI`, `propertyURI` and `maxResults` are replaced by actual values.

Types of requests to the Sindice search service are described in Section 2.4 Search Methods. Sindice distinguishes three types of search: *term search*, *advanced search* and *combined search*. Term search is used when searching for a resource or a literal, advanced search is other cases. En example query is demonstrated on search for the geo microformat in Listing 4.4.

```
* <http://www.w3.org/2006/vcard/ns#latitude> '50.088' AND
* <http://www.w3.org/2006/vcard/ns#longitude> '14.403'
```

Listing 4.4: Example of the geo microformat search query

Obviously, the search is not accurate in that it does not guarantee that the latitude and longitude belong to the same microformat. This is currently a Sindice limitation concerning microformat searches.

4.5 Problems and Suggested Solutions

Several problems emerged during development and testing. In this section, we discuss some of them and suggest solutions.

4.5.1 Content Negotiation

The first problem concerns format of search results returned from Sindice. Semantic Navigator's aim is to enhance navigation on the Web for ordinary users. Search results are to be opened in a web browser and therefore they should be (X)HTML documents. The Sindice API makes it possible to limit search results to a particular format such as RDFa or microformats. However, because of HTTP content negotiation, the Sindice service is unable to discover some (X)HTML documents.

Let us consider the following example: Sindice crawler requests documents with an Accept header similar to `application/rdf+xml`... [29]. A server feeds it a pure RDF document while for a request from a web browser, with a `text/html`... Accept header, it would return an (X)HTML page with embedded RDFa. Thus although there is a human-readable (X)HTML document, which we would like to be able to discover from Semantic Navigator, Sindice does not know about its existence.

Requesting all documents with and without content negotiation, to see if the response is different, would impose too much extra crawling load on websites. One possible solution would be to try it only for a certain number of pages whenever a new site is entered. Then the crawler would decide whether to continue with both versions of the Accept header or not.

4.5.2 Microformat Search

As discussed above, microformat search is limited by Sindice query capabilities. It is not possible to express a query that would select an entity with multiple properties, e.g. a person named "John Doe" who is a member of W3C. What we can do is only search for documents that contain these two triple patterns, not necessarily related:

```
* <foaf:name> "John Doe"
* <sioc:member_of> <http://www.w3.org/data#W3C>
```

Another example is given in Section 4.4.5.

More expressiveness in this case would be very helpful. Sindice is document-centric and only search keywords and patterns withing a document. However, the Sindice team are aware of the problem and work on a new index that would give the possibility to recognize if two search elements match the same entity.[10]

4.5.3 Human-readable properties

A list of all literals in the RDF model of a web page is available through the statusbar context menu. The list is sometimes polluted with many values uninteresting for the user in regard to search, such as various hash values and IDs. A new predicate that would mark other predicates as human-readable or not human-readable could be very useful in similar situations. Semantic Navigator presently filters these values with a regular expression.

[10] `http://goo.gl/7OCKm`

5. Future Work

This chapter provides some ideas and suggestions how functionality of Semantic Navigator could be extended. Several possible future applications are also listed.

5.1 Improvements

One opportunity for improvement is extending supported semantic data formats. As of now, only Microformats supported by Firefox API are detected. Detection could be extended to other microformats recognized by Sindice: XFN, hReview, hListing, hResume. Also, support for microdata could be included.

RDF permits plain literals to be annotated with a (natural) language. Language is specified with `xml:lang` attribute in formats based on XML. Semantic Navigator could let the user select languages he or she understands and display only literals in these languages.

Section 4.5.2 describes a limitation in Sindice query capabilities when selecting an entity with multiple properties. This limitation applies not only to microformats but also to blank nodes in RDF. Semantic Navigator could offer search by blank nodes otherwise. Blank nodes would be identified by some of their properties, e.g. by available inverse functional properties.

The Sindice SPARQL endpoint has only been published very recently and the original specification of this project did not take the endpoint into account. However, it brings a wide range of new possibilities. One of many possible applications, that Semantic Navigator could offer, is a search for instances of a selected class. This feature would be useful particularly with domain-specific ontologies, e.g. one could be interested in all instances of a particular class of diseases.

Last but not least, the extension could drop its dependency on Java. The use of a Java component has significant advantages, such as availability of libraries or easier extensibility. However, the use of Java in Mozilla Firefox extensions is not very common and drawbacks include longer startup time and potential problems with firewalls. As we discussed in Section 4.3, no JavaScript libraries with support for ontologies and inference exist. Implementing a reasoner is beyond the scope of this thesis but a specialized tool sufficient for the limited inference tasks in Semantic Navigator could be developed in future.

5.2 Possible Applications

In this section, we consider several potential applications of Semantic Navigator. Some of them require modifications of Semantic Navigator, however, it was designed with these applications in mind.

5.2.1 Alternative to Semantic Browser

In Section 6.1, we discuss Semantic Browser, a tool that provided inspiration for this thesis. In a sense, Semantic Navigator is its generalization. Semantic Browser works over a set of medical articles with medical terms marked up in the text. If these terms were marked up using RDFa, Semantic Navigator can indeed substitute the interface of Semantic Browser.

5.2.2 Wikipedia-like Navigation

One possible application of Semantic Navigator's approach is a navigation scheme similar to links between Wikipedia articles. In the text of these articles, interesting terms with their own dedicated articles are displayed as hyperlinks. The user can easily look up terms he or she does not understand.

Suppose we would like to have articles on our own website interlinked in this fashion. With Semantic Navigator, it is possible to implement a similar feature in a few steps without the need to mark up documents manually. First, we need to identify terms, that can be linked, and assign URIs to them. For instance, we simply use article names and article URIs for this purpose. Second, we need to generate a list of articles in RDF/XML as shown in Listing 5.1.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://www.example.com/article/Article-one">
    <rdf:label>Article one</rdf:label>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.example.com/article/Article-two">
    <rdf:label>Article two</rdf:label>
  </rdf:Description>
  ...
 </rdf:RDF>
```

Listing 5.1: RDF/XML dump of articles `articles.rdf`

Now we link the list of articles defined in Listing 5.1 to each article:

```
<link rel="index" type="application/rdf+xml" href="articles.rdf" />
```

Third, we need to limit search results to documents from our own website. This is supported by Sindice and can be easily implemented in Semantic Navigator.

Now the Wikipedia-like navigation is prepared. When the user comes across a term he or she wants to know more about, pressing a keyboard shortcut will highlight resources that can be searched for. A resource can be selected and articles discussing the selected term can be listed.

This approach has two advantages over hyperlink navigation in Wikipedia. Documents that did not exist at the time of writing of an article can be discovered and one "link" generated by Semantic Navigator may potentially lead to multiple documents the user can choose from.

5.2.3 Searching in Internal Documents

Linked Data commence to be used by companies as a platform for internal data sharing (e.g. by BBC [10]). The idea behind Semantic Navigator could be applied here too. Some major changes to Semantic Navigator would have to be made in this case, for example replacing Sindice with a search service tailored for intranet web of a company. Nevertheless, it would provide a ready easy-to-use interface for navigation and searching.

6. Related Work

This chapter presents an overview of projects with similar purpose. There are several projects that aim at using structured data in web documents for web navigation or other purposes. What makes them different from Semantic Navigator is mainly Semantic Navigator's versatility. Other projects are usually designed for a specific domain and have either limited vocabulary or limited data set they can work with.

6.1 Semantic Browser

Semantic Browser is a tool that enables easier navigation with relationships as opposed to hyperlinks in the PubMed dataset [19]. The purpose of this tool is to enable easy navigation in abstracts of medical articles following relationships between terms that occur in these articles. An ontology of selected medical terms was created beforehand and their occurrences were marked up in the articles. A web interface is available where the user can click on a term (e.g. "melanoma"), choose one of its properties (e.g. "co occurs with", "is result of"), choose a term with the selected relationship and Semantic Browser will search for documents discussing the selected term.

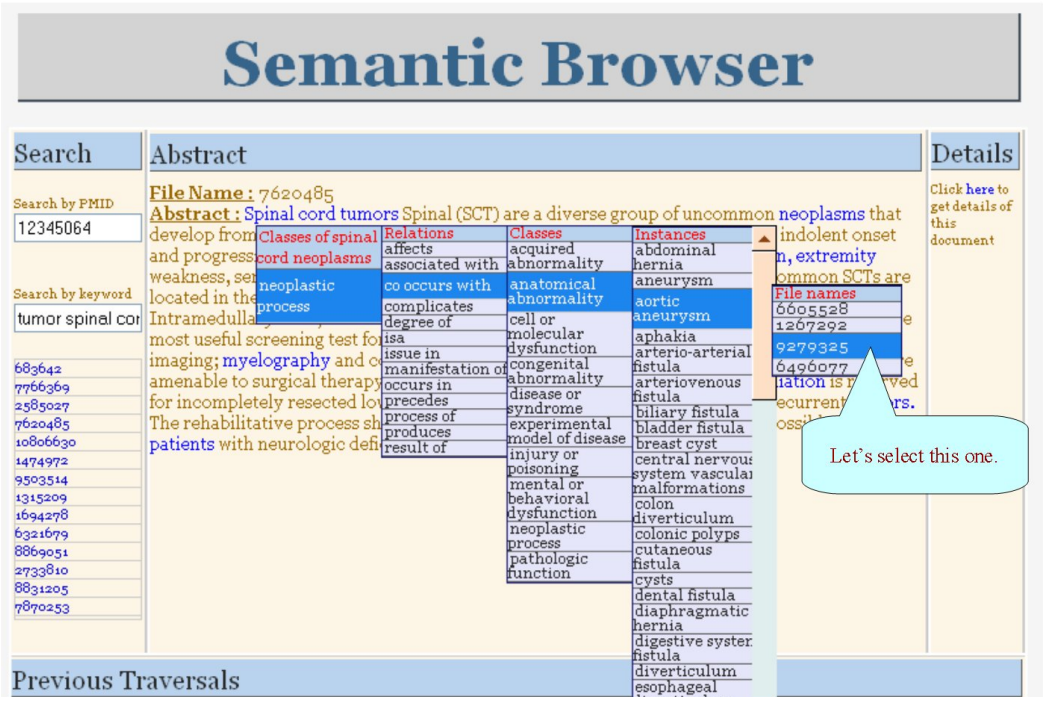


Figure 6.1: Web interface of Semantic Browser
Source: [19]

Semantic Browser provided the initial inspiration for this thesis. The goal of this thesis was to generalize the relationship navigation so that it could be used on the whole World Wide Web and with arbitrary ontologies.

An interesting new project called *Scooner*[1,2], which is based on Semantic Browser, is under development. Given a keyword search, Scooner identifies Linking Open Data (LOD) entity mentions within Yahoo! Boss search result pages. Then it queries the LOD in order to present other related entities. Scooner is implemented as a standalone JavaScript application that runs within a common web browser.

6.2 Magpie

Magpie is an architecture comprising of an Internet Explorer plug-in and servers in role of service providers [17]. Magpie builds upon ontologies and instance data that can be used to annotate semantically unstructured HTML documents. First, the user selects an ontology. Magpie than parses the HTML document and highlights instances from an ontological knowledge base. After clicking on a highlight, the user is offered a list of semantic services that can offer more details about the instance, for example, find values of a property, etc.

Many features of Magpie are common with Semantic Navigator. For instance, the ability to load external semantic data and use it in order to highlight resources in a web document. While Semantic Navigator is dependent on Sindice, Magpie can cooperate with multiple services. Major drawbacks are a need for regularly updated ontologies and instance data or coupling of service providers with the plug-in.

6.3 Mozilla Firefox Extensions

There are several Mozilla Firefox Extensions that work with semantic data embedded in (X)HTML pages.

Operator[3] detects microformats. Various actions are offered for each type of microformat. For example, hCalendar events can be exported, added to Google Calendar, hCard contacts can be looked up in Google Maps and so forth. These semantic actions can be easily added or removed. A similar feature was considered for Semantic Navigator but was rejected because Operator already implements it.

PiggyBank[4] deposits discovered RDF data into local or remote data servers known as *Semantic Banks*.

[1] `http://knoesis.wright.edu/library/tools/scooner/`

[2] `http://wiki.knoesis.org/index.php/Scooner`

[3] `https://addons.mozilla.org/cs/firefox/addon/4106/`

[4] `http://www.openlinksw.com/dataspace/dav/wiki/Main/VOSSemanticBank`

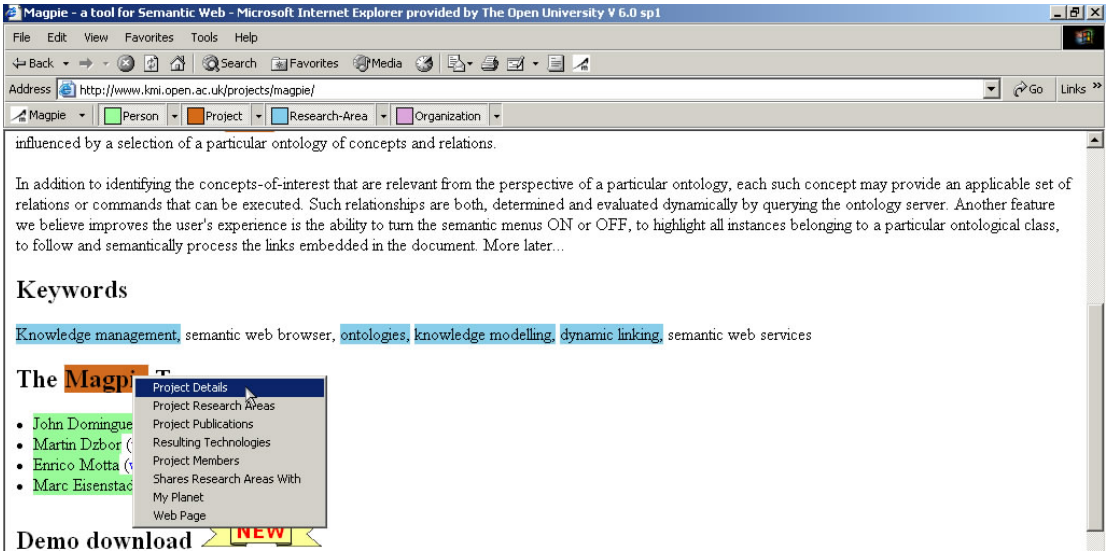


Figure 6.2: Magpie interface. Menu for "Magpie" project lists available services. Clicking on `Project details` opens a new window with details about Magpie.
Source: [17]

Other extensions include *RDFa Developer*[5] or *Semantic Radar* [12] useful mainly for developers. Semantic Radar simply detects RDFa data and optionally pings the Semantic Web Ping Service[6]. RDFa Developer provides features such as RDFa markup examination or SPARQL queries over the RDFa data.

Tabulator[7] provides a human-readable interface for Linked Data. It serves both as a data browser and data editor. Unlike other extensions it does not work with (X)HTML documents but rather with RDF/XML and Notation3 formats.

6.4 Linked Data Browsers

Linked Data Browser allows users to navigate data sources and work with views over RDF data. This represents a different approach to information discovery compared to document-oriented Semantic Navigator. Some examples are listed in [10].

[5] `http://rdfadev.sourceforge.net/`
[6] `http://pingthesemanticweb.com/`
[7] `http://dig.csail.mit.edu/2007/tab/`

7. Conclusion

The aim of this thesis is to make discovery of related web documents easier with the aid of semantic data embedded in web documents together with technologies of the Semantic Web. Semantic web search engines have better understanding of what the user searches for which gives them an advantage over traditional search engines. We present Semantic Navigator, a tool integrated in the Mozilla Firefox web browser as an extension that attempts to bring the advantages of semantic web search to ordinary users.

While other specialized projects that utilize structured data embedded in web documents exist, the contribution of our tool is in its versatility. It is not bound to any specific dataset nor ontology. Semantic Navigator is powered by Sindice, a search engine that provides access to its rich and expanding index of information sources in the Semantic Web.

The goals of this thesis have been accomplished – Semantic Navigator is a working tool that enables users to search for documents with information about an entity or about its properties. The practical application of Semantic Navigator largely depends on two factors – relevance of search results returned by Sindice and availability, quantity and quality of published documents with embedded semantic data. Documents with semantic data currently constitute only a fraction of data published on the Web [23]. Nevertheless, a lot of progress has been made, the amount of semantically enabled sources constantly grows and Google supports adoption of semantic markup with introduction of rich snippets.

As a whole, even though the practical usability of Semantic Navigator is limited due to the low number of web documents with RDFa or microformats markup, the extension successfully demonstrates how advantages of Linked Data can be applied to the traditional web of human-readable documents.

Bibliography

[1] Ben Adida, Mark Birbeck and Steven Pemberton. HTML+RDFa 1.1. Working draft, W3C, 2010.
http://www.w3.org/TR/2010/WD-rdfa-in-html-20100624/.

[2] Ben Adida et al. RDFa in XHTML: Syntax and Processing. Recommendation, W3C, 2008.
http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014/.

[3] David Beckett. RDF/XML Syntax Specification (Revised). Recommendation, W3C, 2004.
http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[4] David Beckett and Tim Berners-Lee. Turtle – Terse RDF Triple Language. Team submission, W3C, 2008.
http://www.w3.org/TeamSubmission/turtle/.

[5] Tim Berners-Lee. Notation 3 (N3) A readable RDF syntax, 1998.
http://www.w3.org/DesignIssues/Notation3.html.

[6] Tim Berners-Lee. Linked data – design issues, 2006.
http://www.w3.org/DesignIssues/LinkedData.html.

[7] Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper San Francisco, 1st edition, 1999.

[8] Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The Past, Present and Future of the World Wide Web by Its Inventor*. Texere, 2000.

[9] Tim Berners-Lee, James Hendler, Ola Lassila et al. The Semantic Web. *Scientific american*, 284(5):28–37, 2001.

[10] Christian Bizer, Tom Heath and Tim Berners-Lee. Linked Data – The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009.

[11] Christian Bizer, Anja Jentzch and Richard Cyganiak. State of the LOD Cloud.
http://www4.wiwiss.fu-berlin.de/lodcloud/state/.

[12] Uldis Bojars, Alexandre Passant et al. An Architecture to Discover and Query Decentralized RDF Data. In *Proceedings of the 3rd Workshop on Scripting for the Semantic Web (SFSW 2007), Innsbruck, Austria*. Citeseer, 2007.

[13] Dan Brickley and Ramanathan V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. Recommendation, W3C, 2004.
http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[14] Gong Cheng and Yuzhong Qu. Searching Linked Objects with Falcons: Approach, Implementation and Evaluation. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):50–71, 2009.

[15] Dan Connolly. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). Recommendation, W3C, 2007.
`http://www.w3.org/TR/2007/REC-grddl-20070911/`.

[16] Mathieu d'Aquin, Enrico Motta et al. Toward a New Generation of Semantic Web Applications. *IEEE Intelligent Systems*, 23(3), 2008.

[17] Martin Dzbor, John Domingue and Enrico Motta. Magpie – Towards a Semantic Web Browser. In *The Semantic Web - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 690–705. Springer, 2003.

[18] Tantek Çelik and Knowledge@Wharton. What's the Next Big Thing on the Web? It May Be a Small, Simple Thing – Microformats, 2005.
`http://knowledge.wharton.upenn.edu/article.cfm?articleid=1247`.

[19] Bilal Gonen. Semantic Browser. Master's thesis, Sakarya University, 2002.

[20] Andreas Harth, Aidan Hogan, Jürgen Umbrich and Stefan Decker. Towards a scalable search and query engine for the web. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 1301–1302. ACM, 2007.

[21] Andreas Harth, Aidan Hogan, Jürgen Umbrich and Stefan Decker. Building a Semantic Web Search Engine: Challenges and Solutions. In *Proceedings of the 3rd XTech Conference*, volume 2008. Citeseer, 2008.

[22] Li Ding Li, Rong Pan et al. Finding and Ranking Knowledge on the Semantic Web. In *The Semantic Web - ISWC 2005*, volume 3729 of *Lecture Notes in Computer Science*, pages 156–170. Springer Berlin / Heidelberg, November 2005.

[23] Richard MacManus. Google's Semantic Web Push: Rich Snippets Usage Growing, June 2005.
`http://goo.gl/CKsGC`.

[24] Frank Manola and Eric Miller. RDF Primer. Recommendation, W3C, 2004.
`http://www.w3.org/TR/2004/REC-rdf-primer-20040210/`.

[25] Martin Mareš and Petr Jankovský. Vyhledávání v textu, 2010.
`http://mj.ucw.cz/vyuka/0910/ads2/6-kmp.pdf`.

[26] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language. Recommendation, W3C, 2004.
`http://www.w3.org/TR/2004/REC-owl-features-20040210/`.

[27] Eyal Oren, Renaud Delbru et al. Sindice. com: A Document-oriented Lookup Index for Open Linked Data. *International Journal of Metadata, Semantics and Ontologies*, 3(1):37–52, 2008.

[28] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. Recommendation, W3C, 2008.
`http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`.

[29] Sindice. *Publishing Web Data*.
`http://sindice.com/developers/publishing`.

[30] Giovanni Tummarello. Sindice, its startup company and 12 billion+ live triples sparql endpoint, June 2011.
`http://blog.sindice.com/2011/06/14/`.

[31] Giovanni Tummarello, Richard Cyganiak et al. Sig.ma: Live views on the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):355 – 364, 2010. Semantic Web Challenge 2009; User Interaction in Semantic Web research.

List of Figures

List of Listings

A. Contents of CD-ROM

The CD attached to this thesis contains the following files and directories:

thesis.pdf
: Electronic version of this thesis.

semantic-navigator-1.0.xpi
: Installation package of Semantic Navigator. How to install the extension is described in Appendix B.

sources
: Source files of the project. See Appendix C for more information about the source files.

B. Installation

Semantic Navigator is an extension for the Mozilla Firefox web browser. The extension can be installed by opening file `semantic-navigator-0.1.xpi` in the browser and confirming the installation dialog. More information about Mozilla Firefox extensions can be found in the official documentation.[1]

System Requirements

The following is required to install and run Semantic Navigator:

- Mozilla Firefox 3.5-5.0
- Installed Java Plug-in version 6 or later[2]

[1] `http://support.mozilla.com/en-US/kb/Using%20extensions%20with%20Firefox`
[2] `http://www.java.com/en/download/`

C. Building the Extension

The extension can be built from sources using Apache Ant. Tools necessary to build Semantic Navigator are:

- Java SE 6 JDK
- Apache Ant[1] version 1.8

Make sure Apache Ant is installed and included in PATH. In order to create installation file of the extension (an `.xpi` file), go to the directory with buildfile `build.xml` and execute:

```
ant clean xpi
```

This will generate the installation `.xpi` file. It is a zip archive containing the actual sources of the extension and packaged Java libraries. Because Any23 and Jena libraries together with their dependencies have over 40 MB, only the necessary parts (repackaged using JBoss Tattletale and libreduce) are included in the `.xpi` file.

The build process can be customized with properties defined in `build.properties`. Some interesting properties are:

use-chrome-jar
: If set to "on" (default), extension sources will be packaged inside a `.jar` archive in order to improve performance.

initial-cache
: If set to "on", an initial ontology cache is included in the installation package. This significantly speeds up Semantic Navigator when used for the first few times.

rebuild-libraries
: If set to "on", Java libraries are repackaged as described above during every build.

Source files overview

The directory where source files are located contains an Apache Ant buildfile and a property file and the following directories:

firefox	Contains extension sources (mostly written in JavaScript and XUL) and resource files such as images
java-lib	Contains Any23 and Jena libraries together with their dependencies
java-src	Contains sources of the Java component used by the extension
tools	Contains tools used to repackage Java libraries and bootstrap Java in the extension

[1] `http://ant.apache.org/`